Johnny 1.01

Simulation
d'un ordinateur Von Neumann simplifié

Peter Dauscher, 2009-2014







Manuel d'utilisation

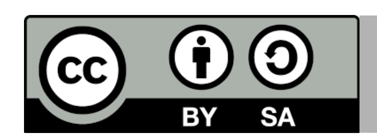


Table des matières

1. Préface

Ce manuel en français est la traduction de la version anglaise écrite par Peter Dauscher. Toutefois, quelques modifications minimes ont été apportées par le traducteur dans l'espoir de rendre les explications plus claires et précises.

Traduction française : Patrice Huaulmé, avril 2015.

2. Introduction

De nos jours, l'architecture Von Neumann est toujours employée dans les ordinateurs, même si par le fait des différentes optimisations successives, leur conception s'écarte de ce modèle initial inventé dans les années 1940. En raison de la miniaturisation amenée par la microélectronique, une grande partie du fonctionnement interne des ordinateurs modernes est cachée aux utilisateurs et est quasiment impossible à observer. Dès lors, la simulation représente une alternative parfaitement adaptée pour donner un aperçu détaillé de ce qui se passe à l'intérieur des ordinateurs.

JOHNNY, un simulateur de processeur, a été développé spécifiquement à des fins éducatives. Pour cette raison, la modélisation fonctionnement du processeur a été simplifiée par rapport à de réels ordinateurs.

3. Considérations didactiques et simplifications

- Le simulateur permet d'observer le fonctionnement du processeur au niveau de l'exécution des instructions d'un programme. Il permet aussi l'examen du déroulement du microcode du séquenceur qui traduit les instructions en différents ordres. Dans un premier temps, la structure interne du séquenceur peut être masquée pour rendre les choses plus simples.
- Les utilisateurs peuvent programmer le simulateur en utilisant un langage assembleur simplifié. Les instructions peuvent être écrites directement dans la mémoire via une interface graphique permettant ainsi d'éviter les erreurs de syntaxe.
- L'unité arithmétique et logique se compose d'un seul registre agissant comme un accumulateur.
- Le simulateur et son interface graphique utilise le système décimal. Bien que ce ne soit pas réaliste, cela simplifie son utilisation, en particulier pour les débutants qui ne sont pas familiarisés avec le système hexadécimal. Ainsi, pour Johnny, 9+8 égale 17 et non 0x11.
- La valeur des données s'étend de 0 à 19 999; celle des adresses de 0 à 999. Les débordements ne sont pas gérés : 0-1 → 0 et 19999+1 → 19999.
- Le jeu d'instructions est très pauvre et est composé de 10 instructions seulement. Pour des raisons de simplicité, toutes les instructions utilisent l'adressage absolu et ont une taille unique d'un mot. Le code opération est donné par le nombre de milliers ; les 3 derniers chiffres représentent l'adresse (exemple, ADD 42 est codé 02 042).
- Dans les processeurs réels, une micro-instruction active en général plusieurs

signaux de commande des bus. Cette simultanéité rend l'observation du comportement plus difficile. La micro programmation s'en trouve plus complexe du fait du risque potentiel de conflits de bus. Par conséquent, une simplification (peu réaliste) a été retenue : à chacune des micro-instructions, est associé un unique interrupteur qui peut être actionné manuellement par l'utilisateur.

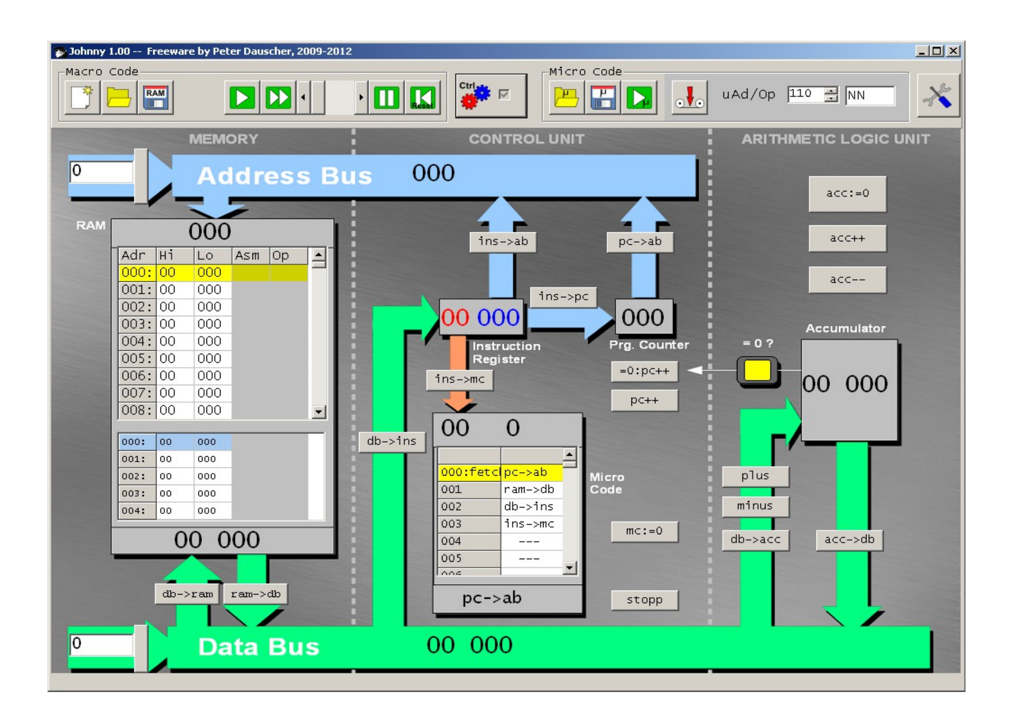
- Une instruction est simplement une suite de micro-instructions. Afin d'éviter l'activation simultanée des commandes d'accès aux bus, le bus de données a la capacité de mémoriser la valeur d'une donnée, ce qui est bien sûr irréaliste : les bus réels sont de simples fils, la rétention est faite par les registres du processeur.

  Le transfert d'une donnée d'un endroit à un autre nécessite deux étapes : a) l'expéditeur place la valeur de la donnée sur le bus, b) le destinataire copie cette valeur présente sur le bus. Ainsi, les conflits de bus sont exclus par construction.

- Le microcode est éditable : l'utilisateur peut créer facilement ses propres instructions en choisissant un nom approprié et en sélectionnant ensuite une suite de micro-instructions avec la souris.

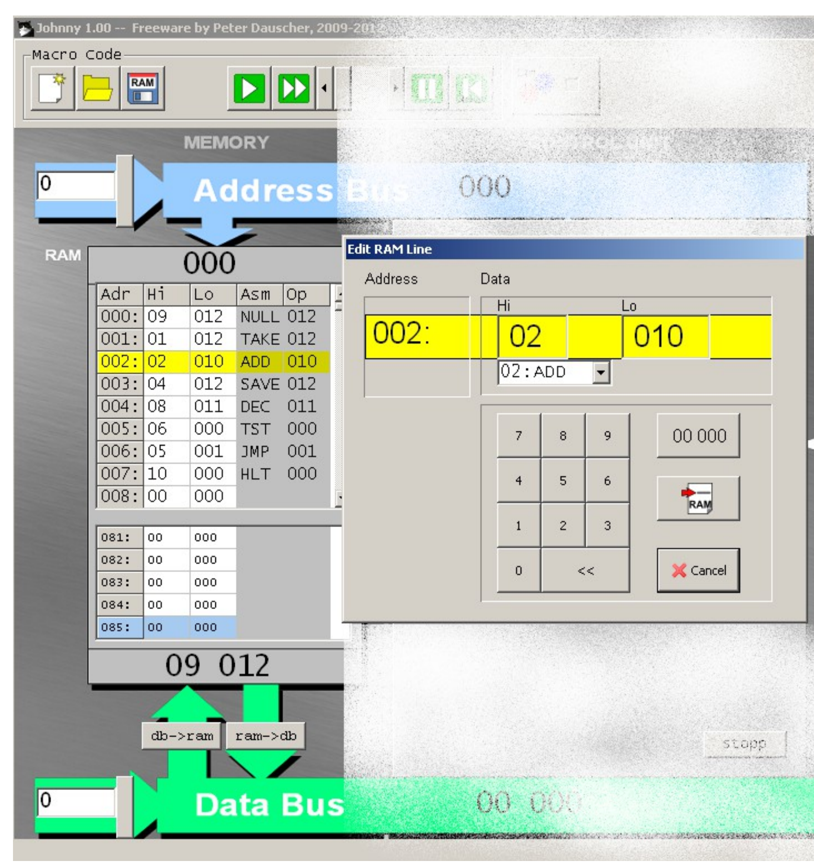
4. Le processeur

Le processeur comprend trois parties : l'unité logique et arithmétique (Arithmetic Logic Unit), la mémoire (Memory) et le séquenceur (Control Unit). Ces blocs sont interconnectés par des bus,



La suite de cette section détaille chaque unité.

4.1. La mémoire (RAM)



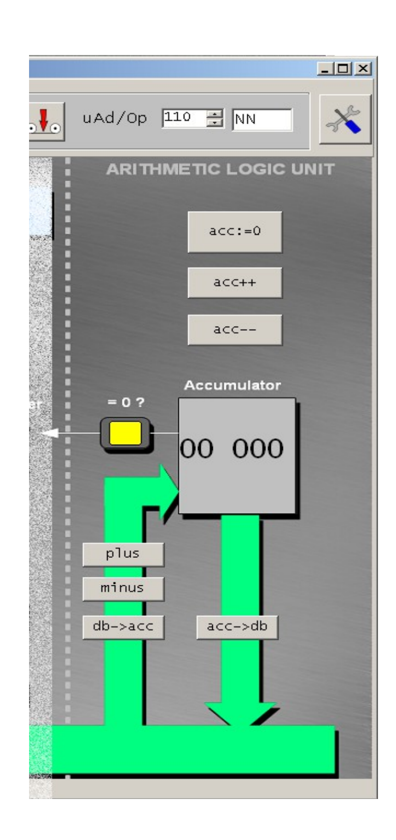
La mémoire à accès direct (RAM : Random Access Memory) comprend 1 000 emplacements, chacun étant capable de stocker un nombre entre 0 et 19 999. Ainsi, un nombre à trois chiffres décimaux suffit à définir une adresse unique pour chaque emplacement.

Pour une facilité de lecture des instructions, le nombre de milliers est séparé des 3 derniers chiffres (colonnes Hi et Lo) car il représente le code opération.

L'utilisation du bouton relatif à la micro-instruction `ram->db` permet de copier le contenu de l'emplacement adressé de la mémoire sur le bus de données; `db->ram` fait le contraire.

Le contenu de la mémoire est éditable par un simple clic de la souris sur un emplacement de la mémoire, qui ouvre alors une fenêtre d'édition. Dans la fenêtre principale, deux portions de la mémoire, qui peuvent se chevaucher, sont affichées. Ainsi, il est possible de voir en même temps les instructions et les données concernées.

4.2. L'unité arithmétique et logique



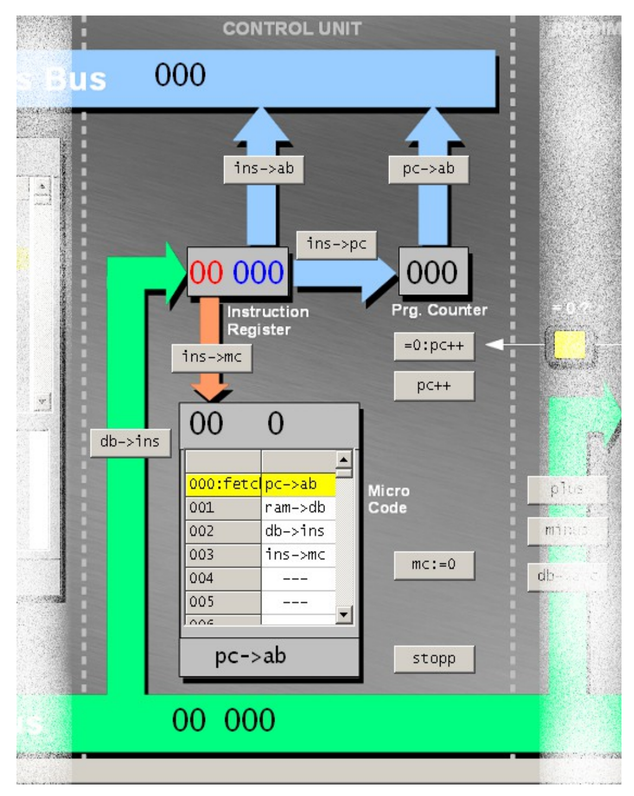
L'unité arithmétique et logique (ALU : Arithmetic Logic Unit) est constituée principalement d'un accumulateur. Ce dernier peut être remis à zéro (`acc:=0`), incrémenté (`acc++`) ou décrémenté (`acc--`).

`db->acc` copie la valeur présente sur le bus de données dans l'accumulateur ; `acc->db` fait le contraire.

La valeur présente sur le bus de données peut être ajoutée (`plus`) ou soustrait (`minus`) à l'accumulateur.

Dans le mode appelé BONSAÏ (voir section 9), quelques unes de ces micro-instructions n'existent pas.

4.3. Le séquenceur



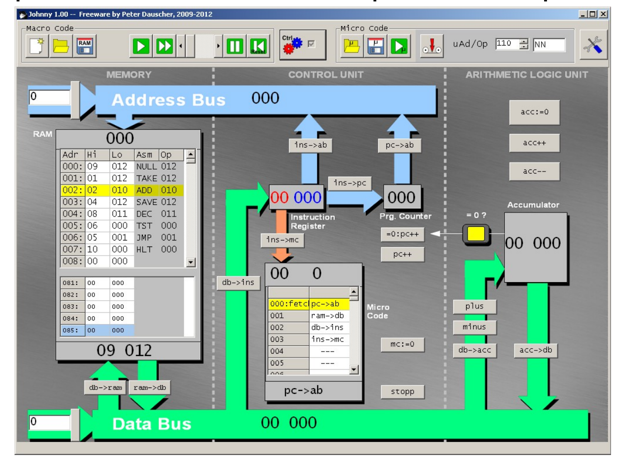
La partie la plus complexe du processeur est le séquenceur (CU : Control Unit). Il comprend le registre d'instruction (IR : Instruction Register), le compteur ordinal (CP : Program Counter) et le microcode.

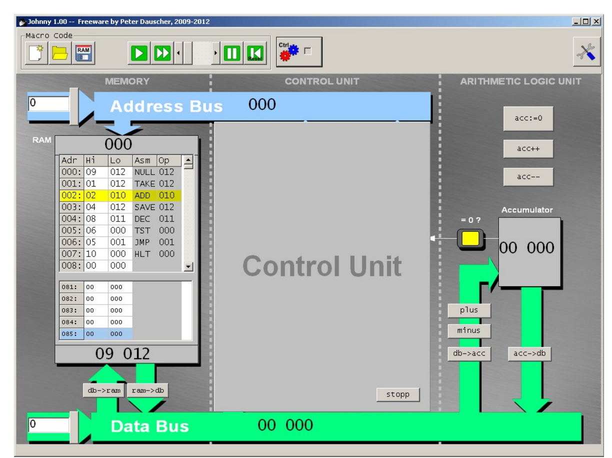
Avec `db->ins` , la valeur présente sur le bus de données est copiée dans le registre d'instruction. La partie de l'instruction qui représente l'adresse peut être copiée directement sur le bus d'adresses (`ins->ab)` ou copiée dans le compteur ordinal (`ins->pc`) pour réaliser par exemple une instruction de saut JUMP. La valeur du compteur ordinal peut être copiée sur le bus d'adresses avec `pc->ab`.

La micro-instruction `pc++` incrémente le compteur ordinal ; `=0:pc++` fait la même chose, mais seulement lorsque la valeur de l'accumulateur est nulle.

`ins->mc` copie la valeur du code opération du registre d'instruction dans l'emplacement des centaines et des dizaines du compteur du microcode (placé au-dessus de la mémoire du microcode) et met à zéro la valeur des unités de ce compteur. La micro instruction `mc:=0` met à zéro la valeur du compteur du microcode ; `stopp` n'est pas une instruction à proprement parler : cela oblige seulement le simulateur à afficher un signalement de fin de programme.

Comme mentionné auparavant, le séquenceur peut être masqué pour rendre les choses plus abordables dans un premier temps.





5. Jeu d'instructions

Le microcode fourni contient les micro-programmes pour les 10 instructions suivantes :

- `TAKE` La valeur de l'emplacement pointé par l'adresse absolue est copiée dans l'accumulateur.
- `SAVE` La valeur de l'accumulateur est copiée à l'emplacement pointé par l'adresse absolue.
- `ADD` La valeur de l'emplacement pointé par l'adresse absolue est ajoutée à la valeur de l'accumulateur.
- `SUB` La valeur de l'emplacement pointé par l'adresse absolue est soustraite à la valeur de l'accumulateur.
- `INC` La valeur de l'emplacement pointé par l'adresse absolue est incrémentée.
- `DEC` La valeur de l'emplacement pointé par l'adresse absolue est décrémentée.
- `NULL` La valeur de l'emplacement pointé par l'adresse absolue est mise à zéro.
- `TST` Si et seulement si la valeur de l'emplacement pointé par l'adresse absolue est nulle, l'instruction suivante est ignorée.
  Note : Ceci est un changement majeur par rapport à la version 0.98 du simulateur où la valeur de l'accumulateur devait être testée directement. Ce changement a été fait dans un souci d'uniformité.
- `JMP` Le programme se poursuit à l'adresse fournie.
- `HLT` Le simulateur affiche un message qui signale que l'exécution du programme est terminée.

6. Exemples de programmes simples

6.1. Addition de deux nombres

Le programme suivant additionne 2 nombres rangés aux adresses <10> et <11>, et mémorise le résultat à l'adresse <12> :

```
001:     TAKE     010
002:     ADD      011
003:     SAVE     012
004:     HLT      000
```

6.2. Multiplication de deux nombres

Le programme suivant multiplie 2 nombres rangés aux adresses <10> et <11>, et mémorise le résultat à l'adresse <12>

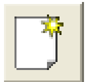
La multiplication de deux nombres est ici mise en œuvre en additionnant de manière répétitive le nombre rangé à l'adresse <10> au résultat à l'adresse <12> dont la valeur initiale est nulle.

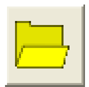
À chaque itération, la valeur stockée à l'adresse <11> est décrémentée. L'itération se poursuit jusqu'à ce que la valeur à l'adresse <11> soit nulle.

```
000:     NULL     012
001:     TAKE     012
002:     ADD      010
003:     SAVE     012
004:     DEC      011
005:     TST      011
006:     JMP      001
007:     HLT      000
```

7. L'interface utilisateur

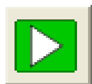
Les boutons sont disposés en deux groupes (code et microcode) dont le dernier s'affiche uniquement si le séquenceur n'est pas masqué. La fonction des boutons relatifs au code est la suivante :

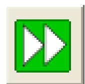
 Met à zéro toute la mémoire RAM (00 000 à chaque emplacement)

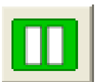
 Charge un programme dans la RAM du simulateur.



Enregistre le programme dans un fichier.

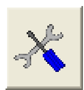
 Exécute l'instruction suivante stockée dans la RAM

 Exécute le programme automatiquement (la vitesse d'exécution est ajustable par le curseur à droite du bouton).

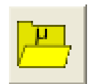
 Arrête l'exécution du programme.



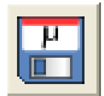
Met à zéro le compteur ordinal et tous les registres du processeur.

 Affiche la fenêtre d'options.

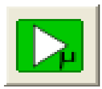
Si le bouton est utilisé pour afficher le détail du séquenceur, les boutons suivants sont alors affichés :



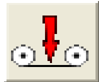
Charge un micro-programme dans la mémoire microcode.



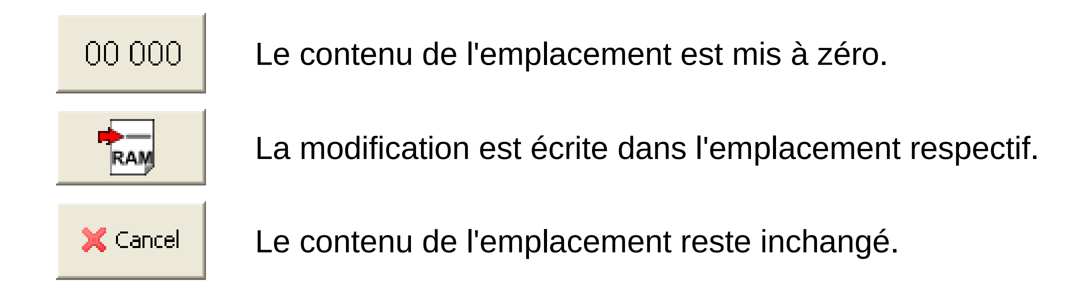
Enregistre le microcode du simulateur dans un fichier.



Exécute uniquement une micro-instruction.

 Enregistre une séquence de micro-instructions afin de définir une nouvelle instruction.

Si vous cliquez sur la valeur d'un emplacement de la mémoire RAM, une fenêtre apparaît dans laquelle la valeur numérique peut être changée ou une instruction peut être choisie depuis une liste déroulante. L'adresse peut être saisie en utilisant le clavier réel ou virtuel. Un double clic sur l'adresse d'un emplacement met à zéro le contenu de cet emplacement.

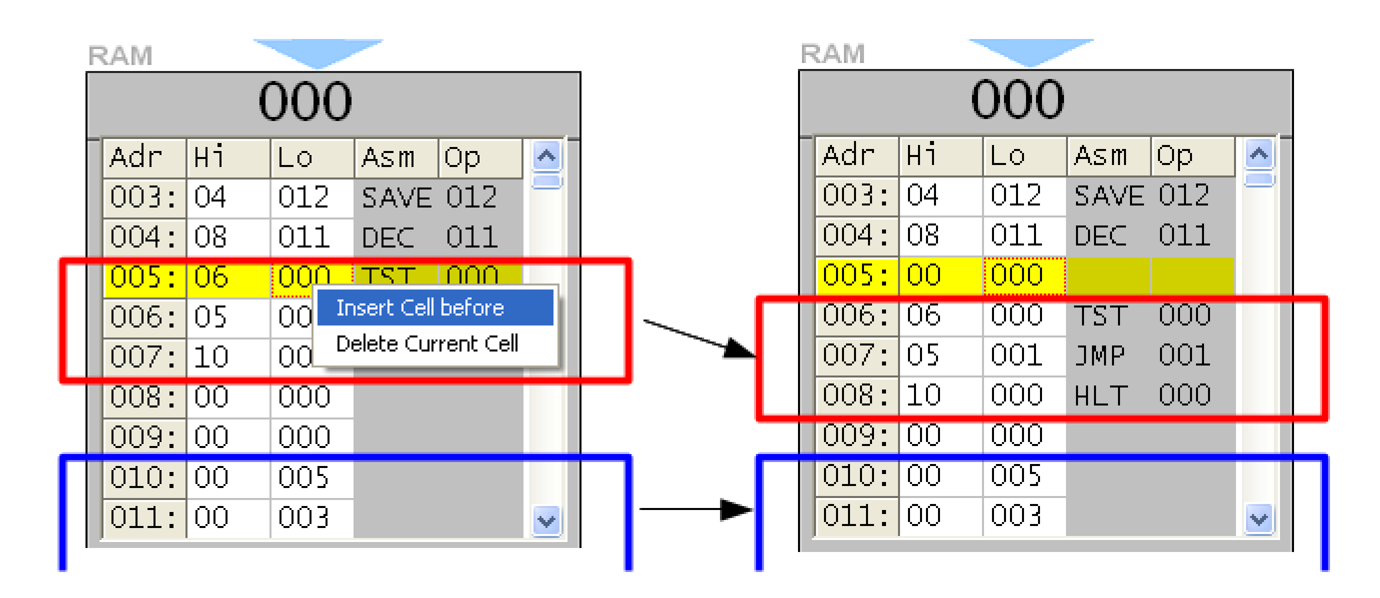


Le contenu de l'emplacement est mis à zéro.

La modification est écrite dans l'emplacement respectif.

Le contenu de l'emplacement reste inchangé.

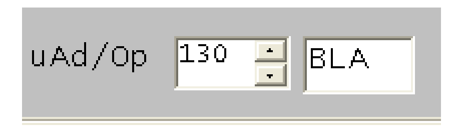
En utilisant le bouton droit de la souris, un menu contextuel est affiché. Ce menu permet des insertions et des suppressions dans la RAM.



Lors d'une opération d'insertion, le bloc formé des données consécutives non nulles est décalé vers le bas d'une case ; la valeur nulle qui suivait ce bloc a été écrasée par cette insertion. De même, lors d'une opération de suppression, le bloc formé des données consécutives non nulles est décalé vers le haut d'une case et une valeur nulle est ajoutée à la fin de ce bloc.

8. Créer sa propre instruction

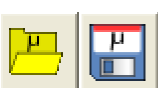
Afin de créer une nouvelle instruction (ou d'en modifier une existante), tout d'abord un code opération et un mnémonique doivent être choisis.



Ensuite, le bouton Enregistrement  est pressé :

La partie respective du microcode est mise à zéro au début. Les appuis sur les boutons relatifs aux micro-instructions sont alors enregistrés. Le clignotement d'une partie de l'interface utilisateur signale l'activation de l'enregistrement. Un appui à nouveau sur le bouton Enregistrement arrête celui-ci.

Le nouveau microcode peut être sauvegardé et puis rechargé ultérieurement avec les boutons suivants :



La sauvegarde du microcode créée deux fichiers : le fichier .mpc qui contient le microcode lui-même, le fichier .nam qui contient les mnémoniques des instructions.

9. Le mode BONSAÏ

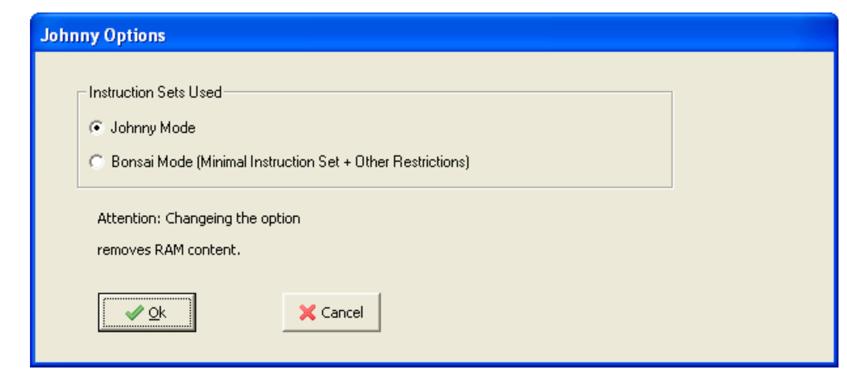
Dans les années 1990, Klaus Merkert et Walter Zimmer mettent en œuvre un simulateur similaire, BONSAÏ. Afin d'être capable d'utiliser le jeu d'instructions BONSAÏ (qui se compose seulement de cinq instructions, à savoir `INC`, `DEC`, `TST`, `JMP` et `HLT`), Johnny peut être commuté dans le mode BONSAÏ.

Le menu déroulant est ensuite adapté à ces instructions, certaines micro-instructions (inutile dans ce mode) sont supprimées.

9.1. Passage en mode Bonsaï

En utilisant le bouton , une fenêtre est affichée, permettant à l'utilisateur de changer de mode.

Attention, le changement de mode efface complètement la RAM.



9.2. Chargement et sauvegarde des programmes

Les programmes de la machine BONSAÏ (fichiers .bma) du simulateur BONSAÏ d'origine peuvent être ouverts et modifiés. Par conséquent, il est possible de créer un programme pour la machine BONSAÏ en utilisant JOHNNY puis de le transférer sur le simulateur d'origine (qui est plus complexe mais aussi plus réaliste).

Il y a donc trois formats de fichiers pour le contenu de la RAM:

.ram	Format standard de contenu de la RAM du simulateur JOHNNY
.bma	Format standard des programmes de la machine BONSAÏ
.bij	Programme Bonsaï enregistré au format standard de JOHNNY

10. Informations légales, techniques et remerciements

10.1. Informations légales



Le programme est un logiciel Open Source[1] et est soumis à la licence GNU GPLv3 : http://www.gnu.org/licenses/gpl-3.0.txt.

La traduction ci_dessous des articles 15 et 16 est fournie dans l'espoir qu'elle facilitera sa compréhension, mais elle ne constitue pas une traduction officielle ou approuvée d'un point de vue juridique.

Article 15. Déclaration d'absence de garantie.

IL N'Y A AUCUNE GARANTIE POUR LE PROGRAMME, DANS LES LIMITES PERMISES PAR LA LOI APPLICABLE. À MOINS QUE CELA NE SOIT ÉTABLI DIFFÉREMMENT PAR ÉCRIT, LES PROPRIÉTAIRES DE DROITS ET/OU LES AUTRES PARTIES FOURNISSENT LE PROGRAMME « EN L'ÉTAT » SANS GARANTIE D'AUCUNE SORTE, QU'ELLE SOIT EXPRIMÉE OU IMPLICITE, CECI COMPRENANT, SANS SE LIMITER À CELLES-CI, LES GARANTIES IMPLICITES DE COMMERCIALISABILITÉ ET D'ADÉQUATION À UN OBJECTIF PARTICULIER. VOUS ASSUMEZ LE RISQUE ENTIER CONCERNANT LA QUALITÉ ET LES PERFORMANCES DU PROGRAMME. DANS L'ÉVENTUALITÉ OÙ LE PROGRAMME S'AVÉRERAIT DÉFECTUEUX, VOUS ASSUMEZ LES COÛTS DE TOUS LES SERVICES, RÉPARATIONS OU CORRECTIONS NÉCESSAIRES.

Article 16. Limitation de responsabilité.

EN AUCUNE AUTRE CIRCONSTANCE QUE CELLES REQUISES PAR LA LOI APPLICABLE OU ACCORDÉES PAR ÉCRIT, UN TITULAIRE DE DROITS SUR LE PROGRAMME, OU TOUT AUTRE PARTIE QUI MODIFIE OU ACHEMINE LE PROGRAMME COMME PERMIS CI-DESSUS, NE PEUT ÊTRE TENU POUR RESPONSABLE ENVERS VOUS POUR LES DOMMAGES, INCLUANT TOUT DOMMAGE GÉNÉRAL, SPÉCIAL, ACCIDENTEL OU INDUIT SURVENANT PAR SUITE DE L'UTILISATION OU DE L'INCAPACITÉ D'UTILISER LE PROGRAMME (Y COMPRIS, SANS SE LIMITER À CELLES-CI, LA PERTE DE DONNÉES OU L'INEXACTITUDE DES DONNÉES RETOURNÉES OU LES PERTES SUBIES PAR VOUS OU DES PARTIES TIERCES OU L'INCAPACITÉ DU PROGRAMME À FONCTIONNER AVEC TOUT AUTRE PROGRAMME), MÊME SI UN TEL TITULAIRE OU TOUTE AUTRE PARTIE A ÉTÉ AVISÉ DE LA POSSIBILITÉ DE TELS DOMMAGES.

La documentation, elle même, est soumise à la licence Creative Commons CC-BY-SA.

http://creativecommons.org/licenses/by-sa/3.0/fr/legalcode

1 NDT : la traduction française « code source ouvert » est peu utilisée

10.2. Informations techniques

Le programme n'a pas besoin d'être installé pour être utilisé. Il peut être exécuté directement depuis un support de stockage.

Le programme a été développé en utilisant l'environnement de développement graphique libre Lazarus (version 0.9.30.4) : http://www.lazarus.freepascal.org. Donc, au moins en théorie, le programme devrait pouvoir être compilé pour tout système d'exploitation que Lazarus est capable de supporter.

Tous les graphiques ont été créés avec OpenOffice, LibreOffice, the Gnu Image Manipulation Program (GIMP) et InkScape :

http://fr.openoffice.org

http://fr.libreoffice.org

http://www.gimp.org

http://www.inkscape.org

10.3. Remerciements

Merci à tous ceux qui ont contribué à créer ce simulateur indirectement par leurs activités dans les projets Open Source mentionnés ci-dessus.

En outre, je tiens à remercier toutes les personnes qui ont fourni des commentaires précieux et des rapports de bogues :

- Mes élèves du Gymnasium am Kaiserdom à Speyer en Allemagne ;
- Mes collègues Jens Fiedler, Ewald Bickelmann, Bernd Fröhlich, Ernst-Lothar Stegmaier et tous les collègues qui ont testés le logiciel avec leur élèves et ont trouvés des bogues ;
- Alexander Güssow et Joachim Brehmer-Moltmann qui ont effectué des tests approfondis ;
- Alexander Domay pour la première compilation sur un système Linux ;
- Klaus Merkert, Tobias Selinger, Martin Oehler et David Meder-Marouelli pour leurs commentaires précieux et leur encouragement pour faire de Johnny un projet open source ;
- Les responsables des conférences Imedia et MNU qui m'ont donné l'occasion de présenter Johnny à un large public ;
- Wolfgang Laun, FH St. Pölten, pour les corrections de ce manuel ;
- En particulier toutes personnes que j'ai oubliées dans la liste.

Je vous souhaite beaucoup de plaisir (et un travail productif) en utilisant Johnny.

Tout rapport de bogue ou autre suggestion est appréciée. Merci d'avance!

Peter Dauscher, le 26 avril 2014

peter.dauscher@gmail.com

